Minirel Project

# BF Layer

This layer heavily relies on LRU list, hash table and free list for the page management.

Predefined variables: BF\_MAX\_BUFS, BF\_HASH\_TBL\_SIZE, BFerrno, all the error codes, …

## void BF\_Init(void)

Initializing the BF layer, with the hash table and the free list.

## int BF\_GetBuf(BFreq bq, PFpage \*\*fpage)

Reading a page from disk.

BF\_GetBuf and BF\_AllocBuf require a free page in the buffer pool. If there is a free page, the head of the free list is popped into this one. Else a victim is chosen to be replaced.

In this implementation, the BF layer is operated using LRU list to select a page as victim.

## int BF\_AllocBuf(BFreq bq, PFpage \*\*fpage)

Allocating memory.

Returns a memory error code in case the memory is full.

## int BF\_UnpinBuf(BFreq bq)

Unpins a page.

Returns the adapted error code if the page is not found in the hash table, not in the buffer pool or already unpinned; BFE\_OK if everything goes as planned.

## int BF\_TouchBuf(BFreq bq)

Marking dirty pages in order to have them written to disk if there are chosen as victims.

Similarly to BF\_UnpinBuf, return an adapted error code, for the same situations.

## int BF\_FlushBuf(int fd)

Writing content to the disk.

## void BF\_ShowBuf(void)

Display the current state of the Buffer pool.

## void BF\_PrintError(const char \*s)

Can be used to print the error log.

BFE\_MISSDIRTY, BFE\_INVALIDTID, BFE\_MSGERR, BFE\_HASHPAGEEXIST unused.

## Other functions:

Added to the 8 mandatory interface routines and the 2 predefined data structures, the buffer pool layer relies on the implementation of LRU-list, Hash-table and Free-list and their operation functions.

### Hash-table for page retrieval, 6 functions:

* An initialization function, allocating memory.
* A clean function to empty the table after it is used.
* An index function (the hashing function) to find a page in the hash table.
* A find function using the index function.
* An add function for new pages.
* A remove function to remove unpinned pages from the table when they are selected as victim by the LRU list.

### LRU, 3 functions:

* A push function to insert new pages in the list.
* A remove function to reorganize the list after a page removal.
* A bigger function to select the least recently used page (unpinned page) and replace it with a new one while saving the ancient one on a free list and sending them to the disk if its dirty.

### Free-list, 3 functions:

* An initialization function.
* A pop function (pop the head of the list).
* A push function (add a new page).

## Possible improvements:

* Concerning the hashtable, the trivial hashfunction  is used. Using a more sophisticated hashing algorithm would be interesting.
* Keeping a pointer on the tail of the LRU to avoid going through the whole list when looking for a victim.

# PF Layer

Predefined variables: PF\_FTAB\_SIZE, PAGE\_SIZE, PFerrno

## void PF\_Init(void):

The PF layer, directly adjacent to the BF layer, rely on the BF layer implantation. Hence, in PF\_Init(void), its initialization begins by calling BF\_Init, while allocating memory area for PF table.

## int PF\_CreateFile (const char \*filename)

With the access() function, the code first checks if the file exists, or wasopened previously, and then create a file, and write initialized header (by

If error occurs, returns predefined error number, and update global variable errno, then print error through PF\_PrintError()

## int PF\_DestroyFile (const char \*filename)

Works similarly to PF\_CreateFile()

## int PF\_OpenFile (const char \*filename)

Open file through syscall open() with check whether file is already opened.

Then, read header value from file with existing file table entries from file.

Finally, returns PF file descriptor.

## int PF\_CloseFile (int fd)

This function closes the file associated with PF file descriptor fd. This implies releasing all the buffer pages belonging to the file from the LRU list to the free list. Meanwhile, dirty pages must be written back to the file if any. All the buffer pages of a file must have been unpinned in order to close successfully the file. If the file header has changed, it is written back to the file. The file is finally closed by using the system call close(). The file table entry corresponding to the file is freed. This function returns PFE\_OK if the operation is successful, an error condition otherwise.

## int PF\_AllocPage (int fd, int \*pageNum, char \*\*pagebuf)

Using given file descriptor, write new page into file and update buffer entries

## int PF\_GetNextPage (int fd, int \*pageNum, char \*\*pagebuf)

Using header of file and given pageNum, the function just iterates through pages until it reaches to pageNum.

## int PF\_GetFirstPage(int fd, int \*pageNum, char \*\*pagebuf)

Simple PF\_GetNextPage(fd,-1,pagebuf)

## int PF\_GetThisPage (int fd, int pageNum, char \*\*pagebuf)

Simple implementation with PF\_GetNextPage with error handling

## int PF\_DirtyPage(int fd, int pageNum)

With iterating through pages, this function checks whether page is dirty and, returns function status

## int PF\_UnpinPage(int fd, int pageNum, int dirty)

With PF\_DirtyPage(), this function checks whether page is dirty, then, free the page memory area, and unpin the page

## void PF\_PrintError (const char \*errString)

Prints error string to stderr, with predefined error code imprinted in front of errString value